LA-UR-81-1082
CONF-810424--3

TITLE: EXPERIENCE WITH A MULTIPROCESSOR BASED ON EIGHT FPS 120B ARRAY PROCESSORS

MASTER

AUTHOR(S): Ingrid Y. Bucher
Paul O. Frederickson
James W. Moore

SUBMITTED TO: FPS Users Group Meeting, St. Louis, April 26-29, 1981

LOS ALAMOS SCIENTIFIC LABORATORY
Post Office Box 1663 Los Alamos, New Mexico 87545
An Affirmative Action/Equal Opportunity Employer

EXPERIENCE WITH A MULTIPROCESSOR BASED ON EIGHT FPS 120B ARRAY PROCESSORS

Ingrid Y. Bucher, Paul O. Frederickson, and James W. Moore

Los Alamos National Laboratory
Los Alamos, NM 87545

ABSTRACT

The rate of increase in the speed of monoprocessors is no longer keeping pace with the needs of our laboratory; accordingly we are investigating the use of parallel processors in large scientific computations. As an initial experiment, a particle-in-cell plasma simulation was adapted to run on a star graph architecture consisting of a UNIVAC 1110 as hub, and up to eight Floating Point Systems AP120B array processors at the other vertices. Subdivision of tasks among processors and measured results are discussed.

INTRODUCTION

A variety of scientific problems at our laboratory could make use of a hundredfold increase in computer power over the next decade [1]. However, it seems unlikely that the new monoprocessors we will see during this time period will be more than 10 times as fast as the fastest available today; thus we expect that the needs of the large-scale users will be met only by machines with true parallel processing capabilities. Current technology interprocessor communication, by means of either direct line or common memory, is a significant factor to be considered in the design of an algorithm to fit on a parallel architecture. Thus our guiding philosophy has been to divide the problem into relatively large tasks with a high degree of independence and, therefore, relatively little interprocessor communication. To avoid common memory contention, we are looking at architectures in which each processor is equipped with a reasonable amount of private memory. In that framework we wish to consider the usefulness of a variety of interconnection schemes.

THE PIC ALGORITHM

Our initial experiment involved a particle-in-cell (PIC) simulation of a plasma [2], which was particularly interesting to study because it exemplifies a class of algorithms that does not effectively utilize the vector capabilities of our fastest computers. In this algorithm the distribution function in position-velocity space that describes the current state of the plasma is represented by several thousand particles. During each timestep the particles are allowed to move through a fine grid approximation to the electromagnetic field. In the simulation we considered, the magnetic field was purely external but the electrostatic field that influenced the motion of the particles was partly caused by the charge carried by the particles themselves.

Accordingly, at the end of every timestep the charge of the particles is computed for each cell of the grid and at the beginning of the next timestep Poisson's equation is solved, giving a potential from which the field can be calculated. These steps are illustrated in Fig. 1a and again in Fig. 1b, the second figure indicating a natural way of dividing the task among p+1 processors. It is clear that the only communication is between processor p_0, which computes the potential from the charge, and the other p processors in which the particles are moved. Thus the algorithm, as we have dissected it, fits rather naturally on any multiprocessor with a star-graph interconnection scheme.

PROGRAMMING THE ARRAY PROCESSOR

Almost all of the computational effort in the algorithm occurs in the relatively small loop in which one particle after another is moved through the field. This is because of the large number of particles that are needed in the simulation to provide a smooth approximation to the distribution function of the plasma. Thus it seemed sensible to spend considerable effort on programming this loop in order to make it as fast as possible. For this reason we chose to code the loop in APAL, rather than FORTRAN.

Our first step was to express the entire loop in terms of basic 120B operations, without deciding exactly when these operations were to be performed. We needed 37 multiplies, 63 floating point adder operations, and 48 references to main data memory.

The next step was to construct a dependency graph of the loop, with one operation at each node. Fig. 2 shows the last few

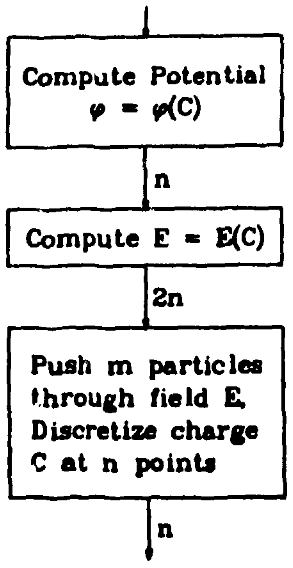
Compute Potential
φ = φ(C)
n
Compute E = E(C)
2n
Push m particles through field E, Discretize charge C at n points
n

Fig. 1a. The time step loop of the PIC algorithm.

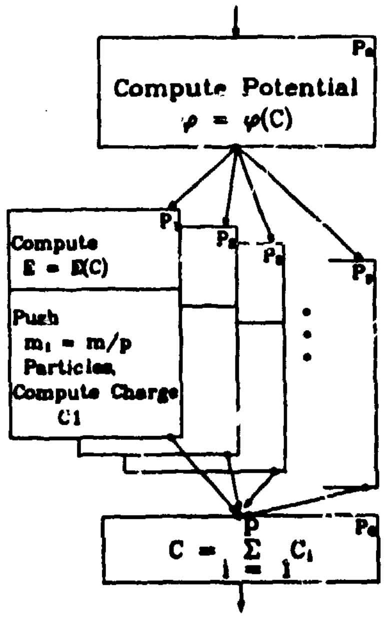

Fig. 1b. One possible multi-processor splitting of this loop.

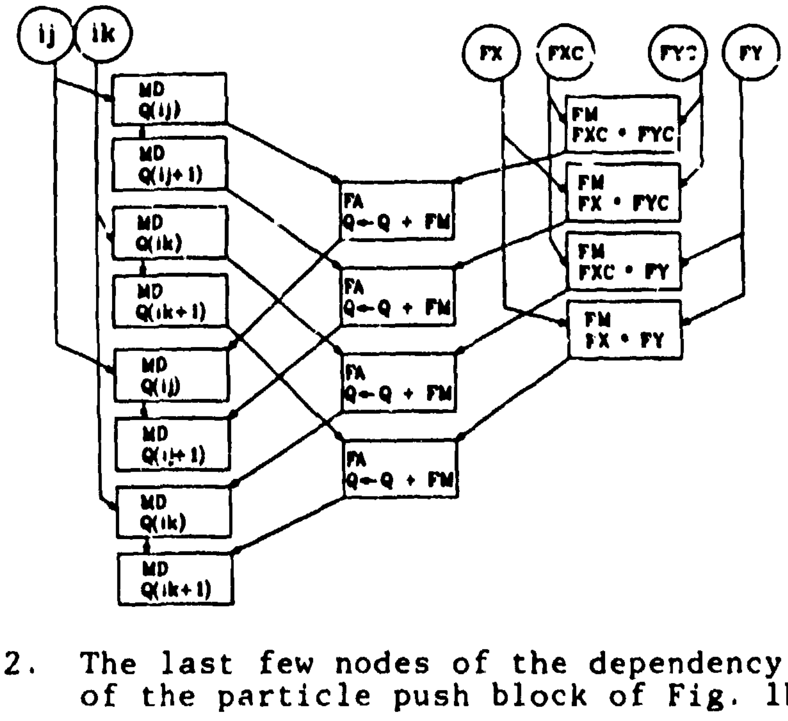

Fig. 2. The last few nodes of the dependency graph of the particle push block of Fig. 1b.

nodes of the graph, in which the charge of one particle is distributed to four cell vertices by piecewise linear interpolation.

The third step in the process was to stretch and adjust the graph to fit into as few cycles as we could, with each line of code containing at most one operation on each functional unit. It was at this stage that we inserted pushes where needed in the graph. In the fourth stage we assigned registers in the x, y and s pads to the temporary variables that were represented earlier by stretched lines in the dependency graph. This required careful bookkeeping, for we had no registers to spare in any of the pads. The small section of APAL code in Fig. 3 corresponds to the section of dependency graph in Fig. 2.

The final stage was to use APSIM to find all the errors we should not have made earlier, but did. In the end we were able to fit the loop into 106 machine cycles, for a computational rate of 5.7 megaflops.

EXPERIMENTAL RESULTS

The Naval Ocean Systems Center in San Diego has nine AP120B array processors attached to a UNIVAC 1110, which is an example of the interconnection scheme that our algorithm requires. The UNIVAC was used as host processor P_0, in which Poisson's equation was solved, and the particle-push vertices were assigned to as many of the APs as could be made available to us.

```
          FMUL DPX(PFX),DPY(FYC)!          "PFX*FYC
             SETMA! ADD# P,IJ                 "GET P(IJ)
"85
          FMUL DPY(PFXC),DPY(FY)!          "PFXC*FY
             INCMA!                           "GET P(IJ+1)
                BEQ DONE!                        "IF L+1 = N, DONE.
                   ADD NX2,IJ                       "IK=NX2+IJ
"86
          FADD FM,MD!                      "P(IJ)=P(IJ)+PFXC*FYC
             FMUL DPX(PFX),DPY(FY)!           "PFX*FY
                SETMA! ADD# P,IK                 "GET P(IK)
"87
          FADD FM,MD!                      "P(IJ+1)=P(IJ+1)+PFY*FYC
             FMUL!                            "PUSH
                INCMA!                           "GET P(IK+1)
                   SUB NX2,IK                       "IJ=IK-NX2
"88
          FADD FM,MD!                      "P(IK)=P(IK)+PFXC*FY
             FMUL!                            "PUSH
                SETMA! ADD# P,IJ! MI<FA          "STORE P(IJ)
"89
          FADD FM,MD!                      "P(IK+1)=P(IK+1)+PFX*FY
             INCMA! MI<FA!                    "STORE P(IJ+1)
                ADD NX2,IJ                       "IK=IJ+NX2
"90
          FADD!                            "PUSH
             SETMA! ADD# P,IK! MI<FA          "STORE P(IK)
"91
          INCMA! MI<FA                     "STORE P(IK+1)
"92
          JMP LOOP                         "RETURN TO TOP OF LOOP

DONE!     FADD FM,MD!                      "P(IJ)=P(IJ)+PFXC*FYC
             FMUL DPX(PFX),DPY(FY)!           "PFX*FY
                SETMA! ADD# P,IK                 "GET P(IK)
"87'
          FADD FM,MD!                      "P(IJ+1)=P(IJ+1)+PFY*FYC
             FMUL!                            "PUSH
                INCMA!                           "GET P(IK+1)
                   SUB NX2,IK                       "IJ=IK-NX2
"88'
          FADD FM,MD!                      "P(IK)=P(IK)+PFYC*FY
             FMUL!                            "PUSH
                SETMA! ADD# P,IJ! MI<FA          "STORE P(IJ)
"89'
          FADD FM,MD!                      "P(IK+1)=P(IK+1)+PFX*FY
             INCMA! MI<FA!                    "STORE P(IJ+1)
                ADD NX2,IJ                       "IK=IJ+NX2
"90'
          FADD!                            "PUSH
             SETMA! ADD# P,IK! MI<FA          "STORE P(IK)
"91'
          INCMA! MI<FA!                    "STORE P(IK+1)
             RETURN                           "RETURN TO MAIN PROGRAM

$END
```

Fig. 3. Reduction to APAL code of the dependency graph segment shown in Fig. 2.

Fig. 4 shows the timing results from a typical run, in which 32,400 particles in 6 array processors are being pushed through a 16 by 16 grid. The computation rate was 34.2 megaflops while all 6 APs were running, as contrasted with about a fifth of a megaflop in the host. We could have achieved a significant speedup by passing the potential calculation to one of the FPS boxes, where the solution would have been computed using a fast fourier transform.

We observed an overhead of between 5 and 13 milliseconds per system call for communication with the array processors or initiation of data transfers. The effect of this overhead is clearly visible in Fig. 4 as a staggering of the start times for the APs, as well as the rather large amount of time required for data communication between host and array processors.

As part of the experiment we rewrote the part of the algorithm destined for the array processors in FORTRAN and compiled that into APAL code. The compiler produced a main loop that required almost 500 cycles to complete. This is about five times as many as the hand-coded version, an efficiency that is not uncommon for the first version of a compiler.

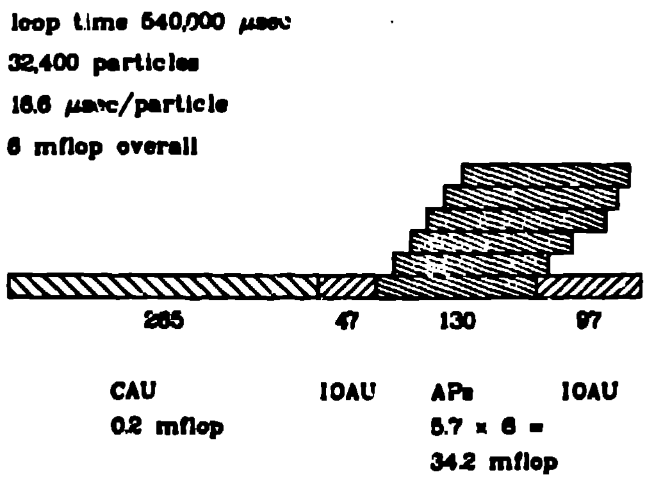

Fig. 4. Timing measurements made on a typical run with the multiprocessor version of the PIC algorithm.

CONCLUSIONS

The results of our experiment lead us to conclude that parallel processing, using the sort of loosely coupled multiprocessor described above, can lead to a significant speedup of certain large scientific problems and particle-in-cell simulations in particular. Our experiment also shows that fast data transfer between processors, or fast access to memory common to processors, is an important design goal of a parallel processor system. In particular, the operating system overhead in interprocessor communication must be minimized.

ACKNOWLEDGMENTS

We would like to thank the staff of the simulation facility at the Naval Ocean Systems Center, particularly Bob Dukelow, John Mayr, and Ron Dahlseid, for their very patient assistance in our experiment.

REFERENCES

1. B. L. Buzbee, W. J. Worlton, G. Michael, and G. Rodrique, "DOE Research in Utilization of High-Performance Computers," Los Alamos Scientific Laboratory report LA-8609-MS (December 1980).

2. R. L. Morse, and C. W. Nielson, "One-, Two-, and Three-Dimensional Numerical Simulation of Two Beam Plasmas," Phys. Rev. Lett. **23**, 1087 (1969).